Recent Advancements of a Genetic Algorithm to Solve the Set Covering Problem (New Aspects of Theoretical Computer Science)

AUTHOR(S):

Iwamura, Kakuzo; Okada, Norio; Deguchi, Yozo

数理解析研究所講究録 1325 巻 2003 年 51-56

Recent Advancements of a Genetic Algorithm to Solve the Set Covering Problem

城西大学数学科

岩村覚三　岡田法雄　出口洋三
Kakuzo Iwamura* Norio Okada† Yozo Deguchi‡

Abstract

A Genetic Algorithm to solve the Set Covering Problem has been proposed by K.Iwamura, T.Sibahara, M.Fushimi and H.Morohoshi[3].Here are shown some recent advancements of the algorithm through applying it to some medium sized input data.

1 Definitions and Domain Specific Knowledge

Let m, n be natural numbers, c_j be positive integers, i.e. *costs*, $(1 \leq j \leq n)$ and Let a_{ij} be 0 or 1 for $1 \leq i \leq m, 1 \leq j \leq n$.

The following Integer Programming Problem,
minimize

$$cx = \sum_{j=1}^{n} c_j x_j \tag{1}$$

subject to

$$\sum_{j=1}^{n} a_{ij} x_j \geq 1 \qquad (1 \leq i \leq m) \tag{2}$$

$$x_j \in \{0, 1\} \tag{3}$$

is called *set covering problem.*

2 A Computational Study

Here, we give a survey of the computational achievements we have so far. We hope that interested readers will consult N. Okada, K.Iwamura and Y. Deguchi[4] and K.Iwamura, M. Horiike and T. Sibahara[5]. Note that the set covering problem is an NP-complete problem.

*Department of Mathematics, Josai University, Japan; kiwamura@math.josai.ac.jp
†Department of Mathematics, Josai University, Japan; nokada@math.josai.ac.jp
‡Department of Mathematics, Josai University, Japan;ydeguchi@math.josai.ac.jp

2.1 Computing Time Dependency on Problem Size

First, we have carried out a computational experiment with Sotec Celeron 400 MHz to see how the Genetic Algorithm with **Method 3** works as the input problem size goes up. Below in Table 1 are the results, where each input problem data was randomly generated with density 3% and uni-cost objective coefficients, i.e. $c_j = 1$ for all j. And still we have set Population size=50, Crossover probability = 0.25, Mutation probability = 0.001, Final generation number = 1000. We can conclude that our Genetic Algorithm runs in proportion to m and the same holds if we fix the number of the rows at 2000 and then vary the number of columns n.

Table 1: Computing Time when m Varies

Problem size	Computing time
200 X 2000	15 seconds
300 X 2000	16 seconds
400 X 2000	17 seconds
500 X 2000	17 seconds
640 X 2000	18 seconds

2.2 Computing Time Dependency on Parameters of the Genetic Algorithm

We have carried out another computational experiment to see parameter dependency of the Genetic Algorithm. We have randomly generated size 640 × 2000 , uni-cost , density 3% set covering input data, which we call 2000N16C. This time we used NEC PC note PC-LM40H32D6 Celeron 400. Computational results appear in Table 2, where * denotes that we have tried 5 trials for the input data 2000N16C. Mean computing time and mean objective function values without * denotes that we have tried 10 trials. The reason we halved the trials was the fact that fluctuations in computing time was very small. For the most fluctuated one with $N = 800$, its computing time were 381,383,398,396,401with fluctuations less than 6%. In the table, N stands for population size in our Genetic Algorithm, p_c: crossover probability, p_m : mutation probability. Comparing the first two lines, we see that letting the final generation number double makes our GA's computing time about two times large with a little bit good objective function values. Comparing the first and the third , we see that changing the values of p_c and p_m doesn't affect our GA's computing time, yet worsens the objective function values. To see how our Genetic Algorithm works when we change N with all other parameters fixed, i.e., $p_c = 0.25, p_m = 0.001$, final generation number= 1000, we have got the results from the fourth line to the last

line. From these results we can say that computing time is almost proportional to N and its objective function values improving a little bit.

Table 2: Parameter Dependency of our Genetic Algorithm

N	p_c	p_m	Final generation number	Objective function value			Mean computing time
				Best	Worst	Mean	
50	0.25	0.001	1000	64	67	65.4	23.6 sec
50	0.25	0.001	2000	64	66	64.2*	*46.2 sec
50	0.50	0.100	1000	77	80	78.7	21.5 sec
25	0.25	0.001	1000	66	70	67.2	11.6 sec
75	0.25	0.001	1000	63	67	65.2	34.7 sec
100	0.25	0.001	1000	63	67	64.9	46.2 sec
200	0.25	0.001	1000	61	66	63.4*	*93.2 sec
400	0.25	0.001	1000	63	66	64.0*	*196.0 sec
800	0.25	0.001	1000	62	65	63.4*	*391.8 sec

2.3 Input Data Dependency of our Genetic Algorithm and a Comparison between our Genetic Algorithm and LINGO 4

For 2 input data in Table 3, computing time of LINGO4 is about 36 to 40 times greater than that of our GA and so we think that as the density of the input problem data grows up, our GA becomes more and more practical in the real world.

In Table 4 are shown the computational results for the input data with

Table 3: Comparison between our GA and LINGO4 for 2 input data of size 999 × 999, density 15% with uni-cost

Prob. Name	GA or LINGO4	VBC(ttime) VFFS(time)	VWC VCFS(time)	VBC/ VCFS
Fd15-Aunicost	GA	21(00:00:50)	23	
	LINGO4	23(00:36:08)	21(00:40:29)	1.000
Fd15-Bunicost	GA	21(00:00:52)	24	
	LINGO4	22(00:32:28)	still 22(00:40:14)	0.955

Table 4: Comparison between our GA and LINGO4 for 4 input data of size 999 × 999, density 1%

Prob. Name	GA or LINGO4	VBC(ttime) VFFS(time)	VWC VCFS(time)	Approximation ratio
2Ed01-Aunicost	GA	162(00:08:52)	165	
	LINGO4	147(00:03:06)	142(00:27:32)	1.141
2Ed01-Bunicost	GA	160(00:09:27)	163	
	LINGO4	153(00:02:13)	144(00:06:46)	1.111
2Ed01UR1to10C	GA	422(00:10:24)	427	
	LINGO4	405(00:01:15)	383(00:36:09)	1.102
2Ed01-Dunicost	GA	168(00:09:17)	170	
	LINGO4	146(00:02:03)	139(00:04:07)	1.209

density 1%. Here, we see that our GA takes much more time than LINGO4 with poorer objective function value and so we see that LINGO4 has defeated our GA.

We have further tested for 5 input data with density 3% in Table 5 and found that for these 3% input data, our GA has recovered its practicality once again. Total approximation ratio for the 13 input problem data, our GA's mean approximation ratio is 1.075 and so our GA is about 8% worse than LINGO4.Yet, we believe that our GA still keeps its practicality for input data with density more than or equal to 3%. As for the 5 input data with density 2% in Table6, differences in computing time between our GA and LINGO4 is not clearcut. And so getting computational results for more bigger input data, say, 2500 rows and 2500 columns with different densities are needed. We will carry out such computational experiments in the succeeding paper.

3 Conclusion

We can say that although our Genetic Algorithm cannot find an optimal solution of the set covering problem, it can find approximate solutions whose objective function values are within about 35% worse than the objective function values LINGO4 finds.

Table 5: Comparison between our GA and LINGO4 for 5 input data of size 999 × 999, density 3%

Prob. Name	GA or LINGO4	VBC(ttime) VFFS(time)	VWC VCFS(time)	Approximation ratio
2Ed03-Aunicost	GA	80(00:01:58)	82	
	LINGO4	71(00:11:42)	still 71(00:40:26)	1.127
2Ed03-Bunicost	GA	77(00:01:53)	79	
	LINGO4	not found(00:40:00)		
2Ed03UR1to10C	GA	114(00:02:07)	116	
	LINGO4	109(00:01:13)	99(00:12:45)	1.152
2Ed03UR5to10D	GA	442(00:01:57)	448	
	LINGO4	not found(00:40:00)		
2Ed03UR5to10E	GA	441(00:01:57)	457	
	LINGO4	424(00:08:21)	423(00:35:24)	1.043

Table 6: Comparison between our GA and LINGO4 for 5 input data of size 999 × 999, density 2% with uni- cost

Prob. Name	GA or LINGO4	VBC(ttime) VFFS(time)	VWC VCFS(time)	Approximation ratio
999X999d2Oct3A02	GA	105(00:03:18)	108	
	LINGO4	94(00:08:09)	92(00:55:19)	1.141
999X999d2Oct3B02	GA	101(00:10:38)	108	
	LINGO4	92(00:07:20)	86(00:25:32)	1.174
999X999d2Oct3C02	GA	103(00:03:11)	107	
	LINGO4	93(00:07:32)	93(01:00:00)	1.108
999X999d2Oct3D02	GA	101(00:06:20)	103	
	LINGO4	94(00:09:31)	94(01:00:00)	1.074
999X999d2Oct3E02	GA	100(00:03:08)	104	
	LINGO4	94(00:09:27)	87(00:20:43)	1.149

References

[1] K.Iwamura[1978], Developing an efficient program code to solve the set partitioning problem, part I(in Japanese),*Abstracts of Spring Conference of the Operations Research Society of Japan(1978)*, pp.107-108.

[2] K.Iwamura and B.Liu[1996],A Genetic Algorithm for chance constrained programming ,*Journal of Information & Optimization Sciences*, vol. 17, no. 2, pp.409-422, 1996.

[3] K.Iwamura, T.Sibahara, M.Fushimi and H.Morohoshi[2000], Set Covering Problem, Genetic Algorithm and Its Domain Specific Knowledge, in *Proceedings of the Second Asia-Pacific Conference on Genetic Algorithms and Applications*, May 3-5, 2000, Global-Link Publishing Company, Hong Kong, 2000,pp.250-257.

[4] N.Okada, K.Iwamura and Y.Deguchi, A Computational Study of a Genetic Algorithm to Solve the Set Covering Problem, presented to *The First International Conference on Information and Management Science*, May 27-31, 2002,Xi'an, China; to appear in *Journal of Interdisciplinary Mathematics.*

[5] K.Iwamura , M.Horiike and T. Sibahara, Input Data Dependency of a Genetic Algorithm to Solve the Set Covering Problem, *Tsinghua Science and Technology*, vol.8, no.1, pp.14-18,2003.